NORMALIZATION REMOVAL AT SCALE: RMSNORM-TO-AFFINE TRANSITION IN GPT-OSS-STYLE MOE TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Although normalization is nearly universal in modern pre-norm Transformers, it is still unclear whether input-dependent normalization statistics are required by the learned computation or if they primarily serve to stabilize optimization. To answer this question, we examined three GPT-OSS-style mixture-of-experts (MoE) transformer variants with 1 billion, 3 billion, and 6 billion total parameters each. These models were trained on 8 billion, 16 billion, and 32 billion processed tokens, respectively, and evaluated on 8 common NLP benchmarks. Our main intervention is a homotopy, or convex combination, from RMSNorm to a scalar affine map at each layer. This combination is parameterized by a coefficient that decreases from 1 (pure RMSNorm) to 0 (pure affine function) during training. The optimal slope and intercept of the affine function were chosen to minimize the mean squared error between RMSNorm and its best linear approximation. Across scales, the normalization-free endpoints retain most of the zero-shot downstream performance of their normalized baselines. Nevertheless, the performance gap becomes more pronounced at larger scales. We also trained learnable element-wise alternatives from step zero. Among the trained variants, the variant initialized as tanh(20x) achieved the best performance, though it remained behind the RMSNorm baseline. The results demonstrate that a simple affine endpoint and a carefully scheduled gate can eliminate the need for normalization statistics. However, the results also demonstrate that the schedule's effectiveness diminishes as the scale increases. Altogether, these results suggest treating normalization as primarily an optimization-stability mechanism and searching for richer, self-normalizing non-linearities that can replace its stabilizing role without the need for explicit, sample-dependent normalization statistics. The code and checkpoints to reproduce this study are available at: https://anonymous.4open.science/r/torchtitan-homotopy-norm-D464/.

1 INTRODUCTION

Normalization layers are a standard component of modern Transformers (Vaswani et al., 2017). In pre-norm architectures such as GPT-OSS (OpenAI, 2025), RMSNorm (Zhang & Sennrich, 2019) is applied before attention and mixture-of-experts (MoE) sublayers, conditioning every residual update. Its computational cost is minimal compared to that of the full model, but its role is structurally significant: it ensures that each sublayer depends on input-dependent activation statistics, while also controlling the scale presented to attention, experts, routers and the output projection.

A natural question is whether this input-dependent operation is part of the essential representational mechanism of the final network or whether it is primarily an optimization tool. With a small increase in validation loss, Baroni et al. (2025) showed that all LayerNorm layers can be removed from GPT-2 models by replacing normalizers sequentially with linear surrogates and fine-tuning between removals. Kanavalau et al. (2026) subsequently introduced TaperNorm, which transforms removal into a continuous gated transition from normalization to a learned sample-independent linear map. Dynamic Tanh (DyT), on the other hand, replaced normalization with an elementwise S-shaped nonlinearity (Zhu et al., 2025). Collectively, these results suggest that the useful effect of normalization may be separable from the specific operation of dividing by a sample-dependent norm.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Our work began from the idea of smoothing the sequential-removal perspective of Baroni et al. (2025) with a continuous homotopy. Since TaperNorm already addresses that broad construction, our contribution is not the convex interpolation itself. Instead, we use TaperNorm as a starting point for a deliberately stripped-down ablation study to determine the minimum amount of machinery needed at a larger pretraining scale. Specifically, we remove its trainable diagonal scale matrices, skip adding an auxiliary scaling term to the loss function, and replace its scaling branch with the scalar affine map $A(x) = aX + b$ that is optimal, under running statistics, for approximating the current RMSNorm output in the least squares sense. The only continuation control is the shared gate g.

We study three GPT-OSS-style MoE variants: Small, Medium, and Large, and vary the gate schedule extensively at Small and Large scale, transferring a selected late schedule to Medium. The experiments partially support the hypothesis: at Small and Medium scale, very late transitions can reach $g = 0$ with modest loss increases; at Large scale, however, there is no comparably broad sweet spot. The final transition becomes fragile, unstable, and schedules that look healthy until the last few steps can fail when the affine endpoint is fully reached.

This scale dependence motivates a second question: perhaps the issue is not only the schedule, but the expressiveness of the endpoint. A scalar affine map is intentionally restrictive. We therefore also study learnable S-shaped pointwise transformations from the start of training. The best completed variant, which was initialized as $\tanh(20x)$, remains worse than RMSNorm but trains stably, suggesting that richer nonlinear yet sample-independent functions are a promising direction beyond $aX + b$.

Our contributions can be summarized as follows:

- A regression-optimal homotopy, $gN(X) + (1 - g)(aX + b)$, whose affine endpoint is determined from four scalar running moments and introduces no new trainable parameters or auxiliary loss.
- A controlled normalization-removal study in GPT-OSS-style MoE Transformers at 1.031B, 3.352B, and 6.090B total parameters, with 0.351B, 0.803B, and 1.232B active parameters and processed-token budgets of 8.39B, 16.78B, and 33.55B.
- A schedule study showing that successful homotopy removal is *late but not abrupt*: very early decays (with respect to the total number of training steps) and near-end drops are unstable.
- Zero-shot evaluation showing that the affine-homotopy endpoints remain systematically weaker overall than normalized baselines, with the gap most visible on the largest model.
- A complementary full-horizon study of S-shaped pointwise replacements, motivating richer self-normalizing nonlinear functions when a scalar affine endpoint is insufficient.

2 Related Work

Normalization and normalization-free optimization. Normalization has long been used to stabilize neural-network optimization, beginning with Batch Normalization, which normalizes activations using mini-batch statistics (Ioffe & Szegedy, 2015); LayerNorm subsequently removed the dependence on mini-batch statistics by normalizing within each example (Ba et al., 2016), while RMSNorm showed that recentering is not necessary for many sequence-modeling settings (Zhang & Sennrich, 2019). In Transformers, normalization is tightly coupled to signal propagation and optimization: mean-field analyses of Pre-LN and Post-LN architectures relate normalization placement to gradient behavior and learning-rate warmup (Xiong et al., 2020), Adaptive model initialization attributes training instability to amplification through residual branches and controls it through adaptive initialization (Liu et al., 2020), and alternatives such as ScaleNorm (Nguyen & Salazar, 2019) and PowerNorm (Shen et al., 2020) modify the statistics used for scale control, while NormFormer (Shleifer et al., 2022) and DeepNorm (Wang et al., 2024) introduce additional normalization, scaling, or residual parameterization to improve optimization at depth. A complementary line of work asks whether the stabilizing role of normalization can instead be built into the architecture or activation dynamics. Fixup trains deep residual networks without normalization through carefully scaled initialization (Zhang et al., 2019), T-Fixup extends this principle to Transformers and removes both LayerNorm and learning-rate warmup (Huang et al., 2020), ReZero initializes residual branches with zero-valued learned gates so that deep networks begin near the identity (Bachlechner et al., 2021), and Normalizer-Free Networks demonstrate at large scale that careful signal-propagation design can substitute for explicit activation normalization (Brock et al., 2021).

Direct normalization removal. The most direct antecedent to our work is normalization removal from an already normalized Transformer. Baroni et al. (2025) show that LayerNorm can be removed from the GPT-2 family by replacing normalization modules one at a time with linearized surrogates and fine-tuning between removal events. Their procedure is sequential and demonstrates that a trained language model can remain useful after its LayerNorm nonlinearities have been eliminated. Kanavalau et al. (2026) make the removal continuous with TaperNorm. For RMSNorm they interpolate between the normalized branch and a sample-independent branch of the form $c\,x D_{\widetilde{\gamma}}$, where c is calibrated from warmup statistics and the per-feature diagonal $D_{\widetilde{\gamma}}$ remains learnable during tapering. They also study explicit output-scale anchoring when removing the final normalization. Their pretraining experiments use small dense models, and they identify larger-scale, deeper, longer-context training as an open direction. Our method is deliberately an ablation of this recipe rather than a new claim about convex interpolation: we remove the learned per-feature endpoint and the normalization-specific auxiliary loss, and replace the endpoint with the online least-squares scalar affine map $aX + b$. This isolates whether an optimal affine approximation, combined with a suitable gate schedule, is sufficient to absorb normalization removal, and tests that hypothesis at 1.031B-6.090B GPT-OSS-style MoEs.

Pointwise and self-normalizing alternatives. A different route is to replace explicit normalization with nonlinearities whose input-output map itself controls scale. Self-Normalizing Neural Networks established this principle outside Transformers using SELU activations whose dynamics drive activations toward a stable mean and variance (Klambauer et al., 2017). More recently, Dynamic Tanh (DyT) replaces Transformer normalization with an elementwise $\gamma \tanh(\alpha x) + \beta$ transformation and reports competitive normalized-versus-norm-free performance across vision and language settings (Zhu et al., 2025). Recent signal-propagation analysis further suggests that tanh-like normalization-free Transformers occupy a different propagation regime and can be more sensitive to initialization and optimization choices (Alekseev, 2026). This line is especially relevant to our Large-scale result: if the best scalar affine predictor $aX + b$ is too weak to replace the stabilizing effect of RMSNorm robustly, a richer sample-independent S-shaped map can provide amplitude compression without per-token normalization statistics. Our $\tanh$ experiments are intended as an initial probe of this design space rather than a claim that one particular nonlinearity is optimal.

3 METHOD

3.1 RMSNORM

For a token representation $X \in \mathbb{R}^d$, the RMSNorm (Zhang & Sennrich, 2019) used in our models is:

$$N(X) = \text{RMSNORM}(X;\gamma) = \frac{X}{r(X)} \odot \gamma, \qquad r(X) = \sqrt{\frac{1}{d}\|X\|_2^2 + \epsilon}, \tag{1}$$

where $\gamma \in \mathbb{R}^d$ is the original learned RMSNorm gain,initialized as an all-one vector. Pre-norm placement is known to materially affect optimization and gradient propagation (Xiong et al., 2020).

3.2 HOMOTOPYNORM

We replace each RMSNorm with a *HomotopyNorm* layer that interpolates between the original normalized output and a scalar affine approximation. Let $N(X)$ denote the original RMSNorm output. We approximate it with:

$$A(X) = aX + b\mathbf{1} \tag{2}$$

where $a, b \in \mathbb{R}$ are shared across all feature coordinates within the normalization module.

The coefficients are chosen to minimize the squared error between the normalized and affine outputs:

$$(a^\star, b^\star) \in \arg\min_{a,b\in\mathbb{R}} \mathbb{E}\Big[\|N(\mathbf{X}) - a\mathbf{X} - b\mathbf{1}\|_2^2\Big] \tag{3}$$

The expectation is over token activations observed by the layer, excluding padding when a mask is provided. Since a and b are scalars, this is ordinary least squares over all unmasked token-feature coordinate pairs. To calculate $a^\star$ and $b^\star$, we therefore maintain the four moments:

$$\mu_X = \mathbb{E}[X], \qquad \mu_N = \mathbb{E}[N(X)], \qquad m_{X^2} = \mathbb{E}[X^2], \qquad m_{XN} = \mathbb{E}[XN(X)] \tag{4}$$

The least-squares normal equations give:

$$a^\star = \frac{m_{XN} - \mu_X \mu_N}{m_{X^2} - \mu_X^2}, \qquad b^\star = \mu_N - a^\star \mu_X \tag{5}$$

In practice, we use the regularized form:

$$a^\star = \frac{m_{XN} - \mu_X \mu_N}{\max(m_{X^2} - \mu_X^2, 0) + \epsilon_{\text{reg}}}, \qquad b^\star = \mu_N - a^\star \mu_X, \qquad \epsilon_{\text{reg}} = 10^{-5} \tag{6}$$

These moments are stored as non-trainable FP32 buffers and updated from each batch using an exponential moving average. After initialization from the first observed batch,

$$m_t = (1 - \rho) m_{t-1} + \rho \, \widehat{m}_t, \qquad \rho = 0.01 \tag{7}$$

where $\widehat{m}_t$ is the corresponding batch statistic computed over all unmasked token-feature coordinates.

Given the resulting affine approximation, HomotopyNorm returns:

$$H(X; g) = gN(X) + (1 - g)\big(a^\star X + b^\star \mathbf{1}\big), \qquad g \in [0, 1] \tag{8}$$

Thus, $g = 1$ exactly recovers the original RMSNorm, whereas $g = 0$ yields the scalar affine map. The gate g is shared across HomotopyNorm modules and is scheduled from 1 to 0 during training.

The current training implementation evaluates $N(X)$ at every forward pass to update the regression statistics, even when g is small. Consequently, the throughput measurements in Section 4 reflect training overhead rather than an optimized $g = 0$ inference path. Functionally, however, the output at $g = 0$ is independent of the normalized branch.

Algorithm 1 in Appendix A gives the method pseudo-code. Figure 1 summarizes the forward computation and online regression procedure used to calculate $a^\star$ and $b^\star$.

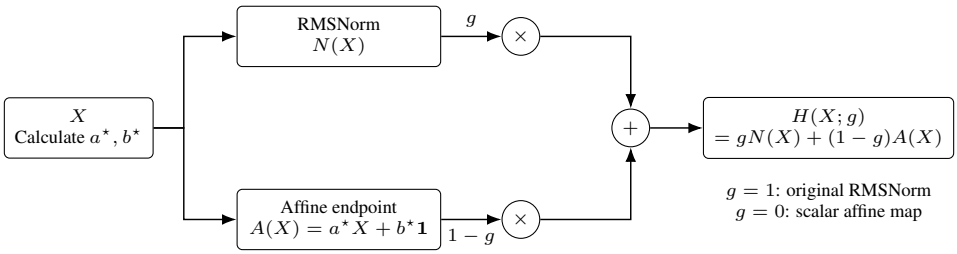


Figure 1: **HomotopyNorm.** Each RMSNorm output $N(X)$ is interpolated with its scalar affine least-squares approximation $A(X) = a^\star X + b^\star \mathbf{1}$. The coefficients are recovered online from exponentially averaged moments of $(X, N(X))$, while the shared gate g is scheduled from 1 to 0.

3.3 Gate schedules

All homotopy results use a single shared gate schedule. Let $u_t = (t - 1)/(T - 1)$ be the normalized training progress. We consider the gate function only on the training interval $t \in [1, T]$. The main schedule family used is of the form:

$$g_{\alpha,\beta}(u_t) = 1 - \max\left(0, \frac{u_t - \alpha}{\beta}\right), \qquad \alpha + \beta = 1, \tag{9}$$

so the model remains fully normalized for the first fraction α of training and reaches $g = 0$ exactly at the final step. We also test polynomial, sinusoidal, and quantized staircase schedules; the complete Small and Large sweeps are reported in Appendix D.

3.4 Richer pointwise endpoints

A scalar affine map can only capture the first-order linear regression of $N(x)$ and cannot reproduce nonlinear scale compression. We therefore study learnable pointwise transformations

$$\phi(x) = \gamma \tanh(\alpha x) + \beta \tag{10}$$

trained from the beginning without explicit normalization statistics and without homotopy schedule. Inspired by Dynamic Tanh (Zhu et al., 2025), our completed runs use several nonlinear initializations from the beginning of training. Section 4 studies their empirical behavior.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

All experiments are implemented in PyTorch using the TorchTitan training library (Liang et al., 2025) and start from its GPT-OSS implementation (OpenAI, 2025). Appendix B gives the exact model/launch configuration used for the reported runs. Appendix C describes the training data and tokenizer construction, Appendix D reports the complete gate-schedule sweeps, Appendix E provides evaluation details and benchmark specifications, and Appendix F reports seed sensitivity experiments and additional checks on training reproducibility.

4.2 HOMOTOPY TRAINING SETUP AND RESULTS

We evaluate normalization removal at three model scales using a single homotopy schedule per scale, selected from the schedule sweeps described in Appendix D. The goal of these experiments is to test whether a normalization-free endpoint can be reached under increasingly higher regimes.

The selected Small-H model uses the late linear 0.94/0.06 schedule. We transfer this schedule to Medium-H, where g remains exactly one until step 15,360 and then decays to zero over the final 640 steps of training. For Large-H, where the transition becomes more sensitive, we use a quantized eight-step staircase schedule:

$$g(u_t) = 1 - \min\left(1, \frac{\left\lfloor 8 \max\left(0, \frac{u_t - 0.96}{0.04}\right)\right\rfloor}{8}\right) \tag{11}$$

This schedule starts from the normalized model at step 31,000, takes values $\{1, 7/8, \ldots, 1/8, 0\}$, and reaches the affine endpoint exactly at the final training step.

Table 1 summarizes training dynamics and runtime characteristics. Throughput, achieved FLOPs, and MFU are reported as medians over the final 1,000 optimizer steps at each scale. Gradient norm and training losses are reported at the final step, peak memory is the maximum reserved memory during training, and validation loss is taken from the final checkpoint.

Table 1: Training and systems summary. Throughput, achieved FLOPs, and MFU are medians over the final 1,000 optimizer steps at each scale.

Run	Tok/s	TFLOP/s/GPU	MFU (%)	Grad norm	Avg. train loss	Max loss	Mem. (GB)	Val. loss
Small	18.72k	58.24	18.67	0.113	2.5667	2.6901	31.99	2.5353
Small-H	17.60k	54.76	17.55	0.320	2.6008	2.7267	31.24	2.5694
Medium	8.51k	61.57	19.73	0.106	2.3446	3.6758	13.36	2.3032
Medium-H	7.95k	57.51	18.43	0.135	2.3911	3.7146	13.50	2.3446
Large	5.13k	55.09	17.66	0.101	2.1552	3.9594	10.74	2.1539
Large-H	4.84k	51.97	16.66	0.595	2.2370	4.0306	10.81	2.2316

The two-branch HomotopyNorm implementation introduces a consistent runtime overhead during the measured late-training window: throughput and MFU decrease by 6.0%, 6.6%, and 5.7% for Small, Medium, and Large, respectively. This overhead comes from continuing to evaluate RMSNorm to update the online regression statistics and should not be interpreted as the cost of an optimized affine $g = 0$ inference path. The quality gap also increases with scale: the final average training-loss difference grows from 0.0341 (2.6008 vs. 2.5667) on Small to 0.0465 (2.3911 vs. 2.3446) on Medium and 0.0818 (2.2370 vs. 2.1552) on Large. The larger final gradient norm of Large-H (0.595 vs. 0.101) is consistent with the increased sensitivity of the late transition observed in the following analysis.

4.3 LATE HOMOTOPY TRANSITIONS ARE SCHEDULE-SENSITIVE

The homotopy transition is successful only when it is both very late and sufficiently gradual. The full schedule sweeps are shown in Appendix D. Here we focus on the best schedules at each scale.

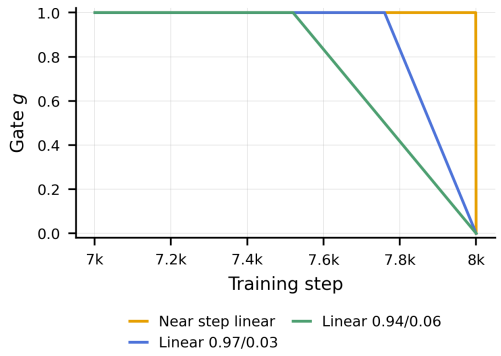
Small. At Small scale, there is a narrow but stable transition window near the end of training. The best schedules are late linear decays: 0.94/0.06 reaches final train/validation losses of 2.60078/2.56943, while 0.96/0.04 reaches 2.60524/2.57780. The selected 0.94/0.06 schedule improves validation loss

by 0.00837 over 0.96/0.04, but remains 0.03411 above the normalized baseline (2.53531). In contrast, schedules that reduce g earlier or more abruptly fail: linear 0.99/0.01, full-training quadratic decay, and a near-final-step drop reach train losses of 5.79848, 6.26328, and 19.16327, respectively. Thus, the transition must be late, but it cannot be arbitrarily compressed.

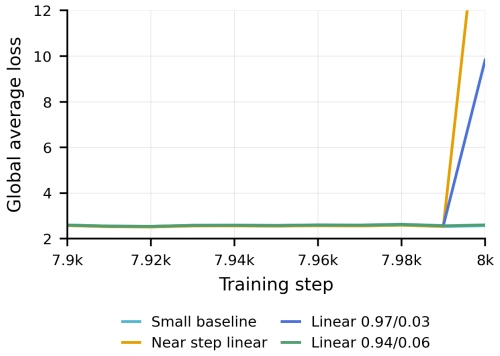
Medium. For Medium, we transfer the 0.96/0.04 schedule without an additional sweep. The model follows the normalized trajectory until step 15,360 and reaches $g = 0$ during the final 640 steps. Medium-H finishes with a training loss of 2.3911 compared with 2.3446 for the normalized baseline, indicating that the same late-transition strategy remains viable at this scale.

Large. At Large scale, schedule sensitivity becomes substantially stronger. The best completed run uses the eight-step staircase schedule and reaches train/validation losses of 2.23697/2.23162, compared with 2.1552/2.1539 for the normalized baseline. Smooth schedules become much less reliable: the 0.94/0.06 linear schedule reaches a train loss of 2.31454, while 0.96/0.04 and 0.99/0.01 degrade to 3.32438 and 7.58551. Several schedules remain close to the baseline until the final steps and diverge only as g approaches zero. This shows that a stable pre-transition trajectory is not sufficient; the final affine endpoint is the critical test.

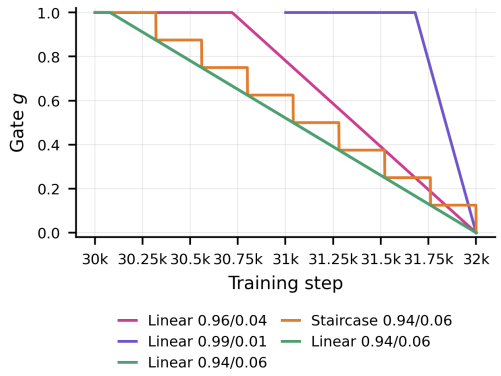
Figure 2 illustrates this behavior. At Small scale, widening the transition window from 1% to 4-6% of training is enough to avoid failure. At Large scale, however, multiple schedules remain stable until the final transition and then diverge sharply, showing that the endpoint becomes increasingly difficult to reach as model size increases. The Large-scale result changes the interpretation of homotopy



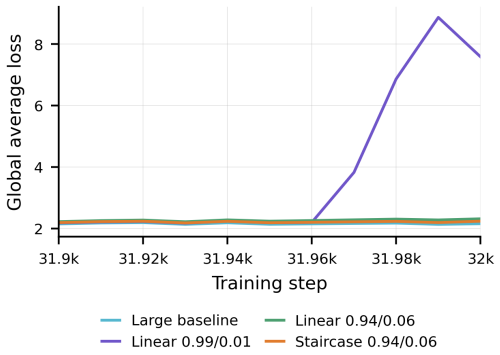
(a) Small: gate schedules g



(b) Small: global average loss



(c) Large: gate schedules g



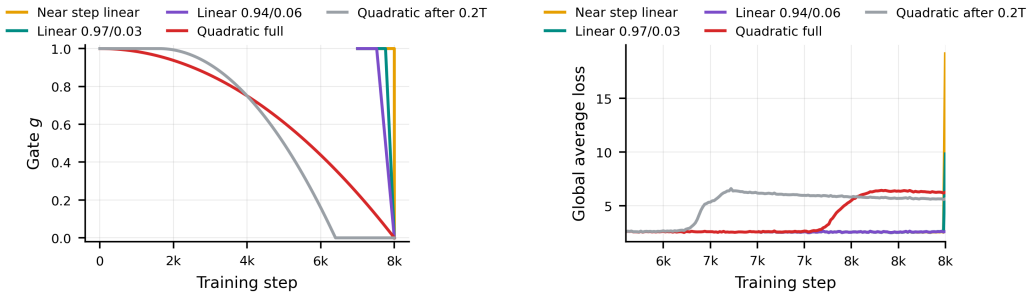
(d) Large: global average loss

Figure 2: **Gate schedules training loss trajectories for representative homotopy ablations.** Left: the shared gate g. Right: the average training loss. On Small, the 0.94/0.06 late-linear schedules remain close and finish stably, whereas the more abrupt 0.99/0.01 schedule blows up only at the very end. On Large, the best result is the eight-step staircase. Schedules can remain near the baseline for most of the terminal window and then spike only when g becomes very small or reaches zero.

removal: while normalization-free endpoints can be reached, the required schedule does not transfer

cleanly across scales. The transition becomes increasingly brittle, so late homotopy should not be viewed as a scale-independent recipe for normalization removal.

Figure 3 further illustrates the main failure cases. Reducing g too early causes instability before training has converged, while reaching $g = 0$ too abruptly causes a terminal loss spike. Even a schedule that behaves well until the final step can fail once the affine endpoint is reached. Conversely, continuing training after reaching $g = 0$ leads to a sustained high-loss regime. In a nutshell, these results indicate a narrow stability window: the transition must happen gradually, and close to the end of training.



(a) Small: gate schedules g

(b) Small: global average loss

Figure 3: **Failure cases of homotopy normalization removal on the Small variant.** Left: representative gate schedules g. Right: the corresponding global average losses. The successful schedule is the late linear 0.94/0.06 run. Three distinct failure modes appear in the ablations. (i) *Too early*: a broad quadratic decay reduces g well before the end of training and triggers an early blow-up. (ii) *Too late*: the linear 0.97/0.03 schedule remains near the stable trajectory until the final point, but still explodes exactly when g reaches zero. (iii) *Too long after the endpoint*: when the model is continued at $g = 0$ for roughly the last 1000 steps, the loss jumps to a high plateau and does not recover.

4.4 Pointwise nonlinear replacements

The homotopy experiments show that a scalar affine endpoint can remove normalization statistics, but the required transition becomes increasingly fragile with scale. We therefore investigate whether a richer sample-independent nonlinear transformation can provide the scale control of RMSNorm without relying on input-dependent statistics. We parameterize the replacement as $\phi(x) = \gamma \tanh(\alpha x) + \beta$, where α, β and γ are learnable parameters. and train the model with this transformation from the beginning, avoiding any homotopy schedule.

We initialize several variants of this pointwise replacement. The simple $\tanh(x)$ initialization fails immediately, reaching a loss plateau around 10 from the beginning of training. In contrast, steeper initializations train successfully. Table 2 summarizes the completed variants. Among these runs, $\tanh(20x)$ performs best, reaching train/validation losses of 2.68020/2.64660 at step 8,000. During training, this model evolves toward a smoother response close to $0.5\tanh(14x) + 0.004$, which we therefore include as an additional initialization. This variant trains successfully but performs slightly worse, reaching 2.73498/2.70031. The $\tanh(10x)$ initialization is intermediate, with losses of 2.68923/2.65481.

Table 2: Completed fixed pointwise alternatives on Small at step 8,000. Lower is better.

Variant	Train loss	Val. loss	Δ val.
Normalized baseline	2.56672	2.53531	0.00000
$\tanh(20x)$	**2.68020**	**2.64660**	+0.11129
$\tanh(10x)$	2.68923	2.65481	+0.11950
$0.5\tanh(14x) + 0.004$	2.73498	2.70031	+0.16500

The normalized baseline remains stronger, with train/validation losses of 2.56672/2.53531. While pointwise nonlinear replacements can train without normalization statistics from the first step, the

tested functions do not yet recover the full performance of RMSNorm. The parameter-count overhead is negligible. $\tanh(20x)$ reaches 18.03k tokens/s, 56.08 TFLOP/s/GPU, and 17.97% MFU, compared with 18.67k tokens/s, 58.07 TFLOP/s/GPU, and 18.61% MFU for the normalized baseline. This is approximately a 3.4% reduction across these measures.

4.5 Downstream evaluation

We evaluate the final checkpoints on seven zero-shot benchmarks and WikiText; the full evaluation protocol and benchmark details are provided in Appendix E. Table 3 reports the primary zero-shot metrics. The homotopy endpoints are close to the normalized model at Small scale but are generally worse, and the gap becomes clearer at Medium and Large scale. On Small, the seven primary accuracy gaps range from 0.05 to 1.57 percentage points. On Medium, they range from 0.39 to 2.90 points. On Large, Large-H is lower on six of seven primary accuracy metrics by 2.12-5.80 points. BoolQ is the sole improvement, from 0.6043 to 0.6153 (+1.10 points).

Table 3: Zero-shot evaluation. ARC-C/E, HellaSwag, and PIQA use normalized accuracy; BoolQ, LAMBADA, and WinoGrande use accuracy. WikiText reports word perplexity (lower is better). Bold marks the better model within each scale pair.

Model	ARC-C ↑	ARC-E ↑	BoolQ ↑	HellaSwag ↑	LAMBADA ↑	PIQA ↑	WinoGrande ↑	WikiText PPL ↓
Small	**0.3294**	**0.5779**	**0.5948**	**0.4464**	**0.3264**	**0.7138**	**0.5193**	**28.7936**
Small-H	0.3242	0.5762	0.5869	0.4358	0.3107	0.7133	0.5185	30.2040
Medium	**0.4053**	**0.6776**	**0.5850**	**0.5699**	**0.4199**	**0.7546**	**0.5627**	**20.2308**
Medium-H	0.3797	0.6486	0.5676	0.5428	0.3947	0.7394	0.5588	21.7839
Large	**0.4539**	**0.7180**	0.6043	**0.6428**	**0.4696**	**0.7845**	**0.5864**	**17.9617**
Large-H	0.3959	0.6818	**0.6153**	0.6005	0.4225	0.7633	0.5493	18.5725

Figure 4 summarizes how the gap changes with model size. The final training-loss gap increases consistently from Small to Large. The validation-loss gap also becomes larger, more than doubling from Small to Large. The average drop across the seven main zero-shot accuracy metrics follows the same trend, increasing from 0.61 percentage points for Small to 3.30 points for Large.

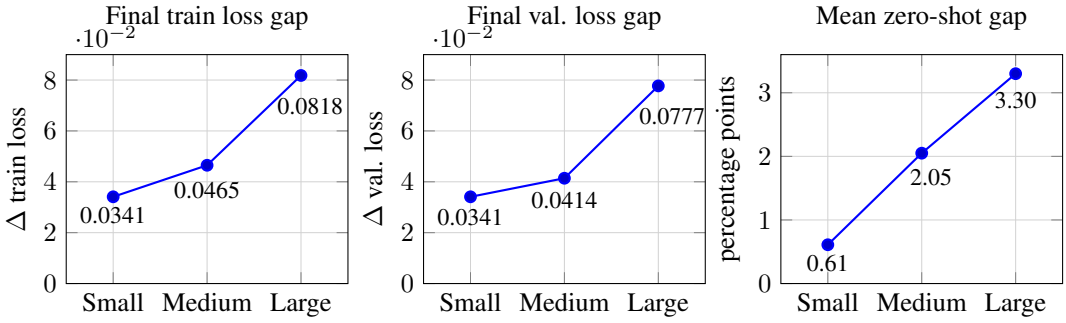


Figure 4: **The normalization-free gap increases with model scale.** From left to right: final training-loss gap, final validation-loss gap, and mean zero-shot accuracy gap between homotopy models and normalized baselines. All gaps increase as the model size grows, showing that reaching the normalization-free endpoint becomes more challenging at larger scales.

5 Discussion

The fact that the studied models can be continued to $g = 0$ supports the hypothesis that much of the role of normalization can be transferred to the surrounding network when the replacement is initialized close to the normalized computation and introduced gradually.

However, this conclusion shouldn't be overstated. The endpoint is not equivalent to RMSNorm, and the downstream results demonstrate persistent quality loss. Furthermore, the Large sweep shows that approximation quality alone does not ensure robustness of continuation: several schedules behave well while $g > 0$ and then spike when the affine branch is fully reached. While a good surrogate and a good schedule are sufficient in some regimes, they do not define a universally applicable solution.

The combined evidence distinguishes between functional necessity and optimization utility. The $g = 0$ homotopy output no longer depends on per-token RMS statistics, and the fixed pointwise models are trained without such statistics from step zero. Nevertheless, the strongest models are the normalized baselines, and the successful homotopies keep g near one for almost all of training. In our experiments, normalization therefore behaves primarily as optimization assistance: it provides a robust scale-control mechanism that makes navigating parameter space much easier.

Calling normalization a training-stability artifact does not mean that it is unimportant. On the contrary, schedule failures demonstrate that its stabilizing effect is significant. The point is that the input-dependent normalization operation itself does not need to be part of the final functional endpoint. What is indispensable is the stabilizing effect it provides while the model is being optimized.

The Small sweep has a clear, late-but-not-abrupt region, whereas the Large sweep has no comparably broad sweet spot. One interpretation is that the scalar affine surrogate is approaching its expressive limit. It captures the best global linear trend between x and $N(x)$, but it cannot reproduce state-dependent compression of activation magnitudes. This suggests a distinction between removing normalization statistics and removing nonlinear scale control. The former is achievable with $aX + b$, while the latter may be the source of brittleness. Pointwise experiments support this view. A steep, bounded function initialized as $tanh(20x)$ can train for the full Small horizon without normalization statistics, though it still trails RMSNorm by 0.11129 validation loss.

Therefore, the next logical step is not a more elaborate schedule for the same affine endpoint but rather a richer class of sample-independent nonlinear functions. We use *self-normalizing* in an operational sense here: the activation itself should compress large excursions, control effective scale, and preserve a stable optimization regime without computing sample-dependent normalization statistics.

Smooth S-shaped functions, such as tanh, are one option. Another attractive option is piecewise-linear S-shaped maps because they can approximate saturation while remaining simple, cheap, and potentially easy to fuse. More generally, one can search for monotone, bounded, or softly saturating pointwise maps whose parameters are initialized from activation statistics but do not require per-token computations during inference. While our completed tanh experiments are not yet competitive with RMSNorm, they suggest that this design space is viable and potentially more scalable than forcing a scalar affine endpoint to absorb the entirety of normalization's nonlinear stabilizing behavior.

6 LIMITATIONS

Our largest model has 6.090B total parameters and 1.232B active parameters, below the released GPT-OSS-20B scale, so the conclusions should not be extrapolated automatically to the full model family. The present HomotopyNorm implementation continues to compute RMSNorm during training in order to update regression statistics, so the reported throughput penalty is not an optimized inference benchmark. The training study uses ClimbMix and a custom 32K tokenizer; behavior may differ under other data distributions, tokenizers, objectives, or post-training pipelines. Finally, these experiments support the interpretation of normalization as primarily optimization scaffolding, but they do not prove that normalization has no other representational effects.

7 CONCLUSION

We test a minimal normalization-removal mechanism across 1.031B–6.090B-parameter GPT-OSS-style MoE Transformers. Each RMSNorm $N(x)$ is continued toward an online least-squares affine approximation $aX + b$ through the homotopy $gN(x) + (1 - g)(aX + b)$. This simple construction can reach viable $g = 0$ endpoints, showing that dependence on input-dependent normalization statistics can be removed. However, the approach has clear limits: successful schedules become increasingly scale-dependent, and the best normalization-free endpoints remain behind RMSNorm in validation and downstream performance.

The results suggest that normalization is primarily valuable as a training-stability mechanism rather than as an essential final computation. Removing its input-dependent statistics is possible, but the stabilizing scale control must be replaced. At larger scales, a scalar affine map is insufficient, motivating richer sample-independent nonlinearities such as smooth or piecewise-linear S-shaped functions. The broader goal is not normalization removal itself, but architectures where stability is built into the activation dynamics rather than provided by a separate normalization operation.

AI USE STATEMENT

Generative AI tools were used to assist with implementation, debugging, summarization of experiment logs, and language editing of the manuscript. All numerical results reported in the paper were taken from experiment logs or evaluation outputs and were checked by the authors. AI-assisted code and prose were reviewed and modified by the authors, who take responsibility for the final content.

ETHICS STATEMENT

This work performs language-model training and evaluation on existing datasets and standard benchmarks and does not involve direct interaction with human subjects.

REPRODUCIBILITY STATEMENT

Model configurations, optimization settings, schedules, seed information, evaluation versions, and the full metric tables needed to interpret the main results are specified in the main text and appendices. Appendix B additionally records the TorchTitan/GPT-OSS implementation, registry variants, shared architectural settings, and the actual launch overrides used for every run. Appendix C records the ClimbMix train/validation provenance and the complete tokenizer-training recipe, including sampling budget, shuffling seed, BPE vocabulary size, minimum frequency, and special tokens.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023. doi: 10.18653/v1/2023.emnlp-main.298.

Sergey Alekseev. Subcritical signal propagation at initialization in normalization-free transformers, 2026.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pp. 1352–1361. PMLR, 2021.

Luca Baroni, Galvin Khara, Joachim Schaeffer, Marat Subkhankulov, and Stefan Heimersheim. Transformers don't need layernorm at inference time: Scaling layernorm removal to gpt-2 xl and the implications for mechanistic interpretability, 2025.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34/05, pp. 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239.

Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1059–1071. PMLR, 2021.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2924–2936, 2019. doi: 10.18653/v1/N19-1300.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-climb: Clustering-based iterative data mixture bootstrapping for language model pre-training, 2025. Accepted to NeurIPS 2025.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2023.

Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4475–4483. PMLR, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

Andrei Kanavalau, Carmen Amo Alonso, and Sanjay Lall. Gated removal of normalization in transformers enables stable training and efficient inference, 2026.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, volume 30, pp. 971–980, 2017.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goeringer, Victor Mustar, François Lagunas, Alexander M. Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 175–184, 2021. doi: 10.18653/v1/2021.emnlp-demo.21.

Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, Sanket Purandare, Gokul Nadathur, and Stratos Idreos. Torchtitan: One-stop pytorch native solution for production ready LLM pretraining. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=SFN6Wm7YBI.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5747–5763. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.463.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Toan Q. Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. In *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong, 2019. Association for Computational Linguistics. URL https://aclanthology.org/2019.iwslt-1.17/.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, 2016. doi: 10.18653/v1/P16-1144.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models, 2023.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34/05, pp. 8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, 2016. doi: 10.18653/v1/P16-1162.

Noam Shazeer. GLU variants improve transformer, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Sheng Shen, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. PowerNorm: Rethinking batch normalization in transformers. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8741–8751. PMLR, 2020.

Sam Shleifer, Jason Weston, and Myle Ott. NormFormer: Improved transformer pretraining with extra normalization. In *International Conference on Learning Representations*, 2022.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. DeepNet: Scaling transformers to 1,000 layers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(10):6761–6774, 2024. doi: 10.1109/TPAMI.2024.3386927.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp. 10524–10533. PMLR, 2020.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019. doi: 10.18653/v1/P19-1472.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1gsz30cKX`.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch FSDP: Experiences on scaling fully sharded data parallel, 2023.

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14901–14911. IEEE, 2025.

A Pseudocode and Algorithm

Algorithm 1 Implementation-equivalent forward pass of HomotopyNorm. Running statistics and g are FP32 non-trainable buffers; $\rho = 0.01$ and $\epsilon_{\text{reg}} = 10^{-5}$.

Require: activation X, gate g, RMSNorm N
$n \leftarrow N(X)$
if training **then**
  flatten the unmasked coordinate pairs (X_i, N_i)
  $\widehat{\mu}_X \leftarrow \text{mean}(X_i)$; $\widehat{\mu}_N \leftarrow \text{mean}(N_i)$
  $\widehat{m}_{X^2} \leftarrow \text{mean}(X_i^2)$; $\widehat{m}_{XN} \leftarrow \text{mean}(X_i N_i)$
  if first statistics update **then**
    $(\mu_X, \mu_N, m_{X^2}, m_{XN}) \leftarrow (\widehat{\mu}_X, \widehat{\mu}_N, \widehat{m}_{X^2}, \widehat{m}_{XN})$
  else
    for $q \in \{\mu_X, \mu_N, m_{X^2}, m_{XN}\}$ **do**
      $q \leftarrow (1-\rho)q + \rho\widehat{q}$
    end for
  end if
  $v_X \leftarrow \max(m_{X^2} - \mu_X^2, 0)$
  $c_{XN} \leftarrow m_{xn} - \mu_X \mu_N$
  $a \leftarrow c_{XN}/(v_X + \epsilon_{\text{reg}})$
  $b \leftarrow \mu_N - a\mu_X$
  return $g\,N + (1-g)(aX + b)$
end if

The statistics update is performed under `no_grad`. During evaluation, the stored moments are reused without modification. In the current training implementation $N(x)$ is still evaluated even at $g = 0$ so that the running regression target remains available during training; this is why the reported homotopy throughput is a training-system measurement rather than an optimized affine-endpoint inference benchmark.

B Implementation Details

The codebase uses TorchTitan (Liang et al., 2025)'s GPT-OSS (OpenAI, 2025) implementation as the base model/training stack. The research model builder is parameterized by a `norm_config_factory`. Baseline registry entries use `RMSNorm`; the `small_homotopy`, `medium_homotopy`, and `large_homotopy` entries use `HomotopyNorm`. This factory is applied to the final model normalization and to the attention and feed-forward normalizations of every Transformer block. `HomotopyNorm` subclasses RMSNorm directly, so the original learned `weight` key is preserved; the gate and regression statistics are additional non-trainable buffers. This makes normalized checkpoints directly reusable as the starting point of late homotopy branches while keeping the original normalization parameters in place. Table 4 summerizes these registry variants.

Table 4: Exact research-model registry variants. The homotopy variants have the same architecture as their normalized counterparts and differ only through the normalization factory. Total/active parameter counts are unchanged up to negligible non-trainable scalar buffers.

Variant	Registry key	Width	Layers	MoE hidden	Experts	Heads/KV	Top-k	Total/active params	Norm factory
Small	`small`	1,024	12	1,536	16	32/4	4	1.031B / 0.351B	RMSNorm
Small-H	`small_h`	1,024	12	1,536	16	32/4	4	1.031B / 0.351B	HomotopyNorm
Medium	`medium`	1,536	18	1,536	24	48/6	4	3.352B / 0.803B	RMSNorm
Medium-H	`medium_h`	1,536	18	1,536	24	48/6	4	3.352B / 0.803B	HomotopyNorm
Large	`large`	1,792	21	1,792	28	56/7	4	6.090B / 1.232B	RMSNorm
Large-H	`large_h`	1,792	21	1,792	28	56/7	4	6.090B / 1.232B	HomotopyNorm

All variants use vocabulary size 32,768, head dimension 64, alternating sliding attention on even-indexed layers, top-4 token-choice routing in the sparse-MoE tradition (Shazeer et al., 2017), softmax router scores with router normalization, `score_before_experts=False`, load-balancing coefficient 10^{-3}, and unfused QKV projections.

The attention layout uses grouped-query attention (Ainslie et al., 2023). Rotary embeddings follow RoPE (Su et al., 2021) with the configured YaRN extension (Peng et al., 2023): RoPE dimension 64, maximum RoPE length 131,072, $\theta = 150{,}000$, $\beta_{\text{slow}} = 32$, $\beta_{\text{fast}} = 1$, and original sequence length 4,096. The expert feed-forward blocks use the GPT-OSS SwiGLU-style design (Shazeer, 2020). Table 5 records the effective settings used to produce the results.

Table 5: Launch settings for the six main runs. All runs use global batch 256, gradient accumulation 1, sequence length 4,096, 2,000 warmup steps, fused AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.1, and BF16 parameters with FP32 reductions.

Run	Peak LR	Steps	Local batch	Global batch	GPUs	Seq.	Initialization/branch	g schedule
Small	6×10^{-4}	8,000	32	256	8	4,096	step 0	–
Small-H	6×10^{-4}	8,000	32	256	8	4,096	normalized ckpt 7000	linear 0.94/0.06
Medium	4×10^{-4}	16,000	8	256	32	4,096	step 0	–
Medium-H	4×10^{-4}	16,000	8	256	32	4,096	normalized ckpt 15,000	linear 0.96/0.04
Large	3×10^{-4}	32,000	4	256	64	4,096	step 0	–
Large-H	3×10^{-4}	32,000	4	256	64	4,096	normalized ckpt 31,000	8-step staircase 0.96/0.04

All three scales use the 32,768-token tokenizer, head dimension 64, top-4 expert routing, sequence length 4,096, and an effective global batch of 256 sequences, corresponding to 1,048,576 processed tokens per optimizer step. Table 6 gives the exact model sizes and budgets.

Table 6: Model and pretraining configurations. "Active" is the parameter count active for one token under top-4 routing.

Model	Total params	Active params	Layers	Width	MoE hidden	Experts	Steps	Tokens	GPUs
SMALL	1.031B	0.351B	12	1,024	1,536	16	8,000	8.39B	8
MEDIUM	3.352B	0.803B	18	1,536	1,536	24	16,000	16.78B	32
LARGE	6.090B	1.232B	21	1,792	1,792	28	32,000	33.55B	64

The exact total/active parameter counts are 1,030,753,856 and 350,686,784 for Small; 3,352,222,992 and 802,524,432 for Medium; and 6,089,999,972 and 1,231,859,300 for Large. Attention head/KV-head counts are $(32, 4)$, $(48, 6)$, and $(56, 7)$ from Small to Large, following grouped-query attention (Ainslie et al., 2023). Per-rank batch sizes are 32, 8, and 4 with gradient accumulation one.

Optimization uses fused AdamW (Loshchilov & Hutter, 2019) with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, and weight decay 0.1. Learning rate follows a cosine annealing schedule, with peak learning rates at 6×10^{-4}, 4×10^{-4}, and 3×10^{-4} for Small, Medium, and Large. Each run uses 2,000 warmup steps followed by cosine decay over the remaining steps to a minimum learning-rate 0.1 times the peak learning rate. Training uses BF16 parameter casting, FP32 reductions, full activation checkpointing, and PyTorch FSDP (Zhao et al., 2023) on NVIDIA A100-SXM4-40GB GPUs. The reported training runs were executed without `torch.compile`. HomotopyNorm uses momentum $\rho = 0.01$ and regression stabilizer $\epsilon_{\text{reg}} = 10^{-5}$. Its regression statistics and gate buffer are stored in FP32. The homotopy runs use the standard cross-entropy objective only: there is no additional scale-anchoring or normalization-specific regularization term.

C DATA AND TOKENIZER

We train on Nemotron-ClimbMix (Diao et al., 2025), using a pre-shuffled copy of the training corpus.Validation uses a held-out split drawn from ClimbMix and excluded from the training set. The original GPT-OSS architecture uses a vocabulary of 201,088 tokens. For the research-scale replicas, we instead train a 32,768 tokens BPE tokenizer on ClimbMix and use that tokenizer for every baseline and homotopy variant. Tokenizer training reads the ClimbMix training split in streaming mode, applies a buffer shuffle, and accumulates non-empty text examples until the configured raw UTF-8 text budget would be exceeded. Table 7 records the exact settings of the tokenizer training.

Table 7: Tokenizer-training configuration used by all research models.

Setting	Value
Tokenizer family	Byte-level BPE (Sennrich et al., 2016)
Training source	ClimbMix training split
Vocabulary size	32,768
Original GPT-OSS vocabulary	201,088
Raw text budget	8,000,000,000 UTF-8 bytes
Shuffle seed	42
Shuffle buffer	100,000 examples
Text batch target	4,000,000 bytes
Minimum token frequency	2
Special tokens	<|bos|>, <|eos|>, <|pad|>, <|unk|>

D HOMOTOPY SCHEDULES

We write $u = (t-1)/(T-1)$ and $[x]_0^1 = \mathrm{clip}(x, 0, 1)$. All train loss values in Table 8 are terminal train losses at step 8,000.

Table 8: Small homotopy schedule sweep. Lower is better.

Schedule	$g(u)$	Timing	Final train loss
Linear 0.94/0.06	$1 - [(u - .94)/.06]_0^1$	final 6%	**2.60078**
Linear 0.96/0.04	$1 - [(u - .96)/.04]_0^1$	final 4%	2.60524
Linear 0.97/0.03	$1 - [(u - .97)/.03]_0^1$	final 3%	9.83374
Linear 0.99/0.01	$1 - [(u - .99)/.01]_0^1$	final 1%	5.79848
Near-step linear	$1 - [(u - .99988)/.00012]_0^1$	effectively last step	19.16327
Quadratic full	$1 - u^2$	entire training	6.26328
Quadratic after $0.2T$	$1 - ([(u - .2)/.8]_0^1)^2$	$0.2T$–T	6.39529
Quadratic 0.2–$0.8T$	$1 - ([(u - .2)/.6]_0^1)^2$	zero at $0.8T$	5.65520
Late quadratic	$1 - ([(u - .95)/.05]_0^1)^2$	final 5%	2.61086
Power-50	$1 - u^{50}$	very late smooth decay	2.61177
$0.5\times$ power-50	$0.5(1 - u^{50})$	starts at $g = 0.5$	3.28714

For the power family define $z(u) = [(u - .94)/.06]_0^1$. All train loss values in Table 9 are terminal train losses at step 32,000.

Table 9: Large homotopy schedule sweep. All listed runs reach $g = 0$ at step 32,000. Lower is better.

Schedule	Shape/window	Final train loss
8-step staircase	$1 - \min\big(1, \lfloor 8 \max\big(0, \frac{u-0.96}{0.04}\big)\rfloor /8\big)$	**2.23697**
Linear 0.91/0.09	$1 - [(u - .91)/.09]_0^1$	6.58604
Linear 0.94/0.06	$1 - [(u - .94)/.06]_0^1$	2.31454
Linear 0.96/0.04	$1 - [(u - .96)/.04]_0^1$	3.32438
Linear 0.99/0.01	$1 - [(u - .99)/.01]_0^1$	7.58551
Power $p = 10$	$1 - z^{10}$	4.14533
Power $p = 20$	$1 - z^{20}$	2.90107
Power $p = 40$	$1 - z^{40}$	3.21712
Power $p = 80$	$1 - z^{80}$	2.82063
Power $p = 320$	$1 - z^{320}$	23.23655
Power $p = 1280$	$1 - z^{1280}$	8.09813
Sinusoidal 0.94/0.06	Linear 0.94/0.06 modulated by sinusoidal decrease	4.75697
Sinusoidal 0.96/0.04	Linear 0.96/0.04 modulated by sinusoidal decrease	6.33612

E EVALUATION

We evaluate the six baseline/homotopy checkpoints zero-shot with ARC-Challenge and ARC-Easy (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), LAMBADA-OpenAI (Paperno et al., 2016), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2020), and WikiText (Merity et al., 2017). ARC-Challenge/Easy, HellaSwag, and PIQA use length-normalized accuracy as the primary multiple-choice metric; BoolQ, LAMBADA, and WinoGrande use accuracy; WikiText uses word perplexity. Evaluation uses `lm-eval==0.4.12` from the EleutherAI Language Model Evaluation Harness (Gao et al., 2023), pinned to commit `6d642546f4688648fced259eb3302efd36ece5af`. Benchmark datasets are fetched through the harness using Hugging Face Datasets (Lhoest et al., 2021). The evaluated checkpoints are step 8,000 for Small/Small-H, 16,000 for Medium/Medium-H, and 32,000 for Large/Large-H. All downstream rows are seed 0.

F SEED SENSITIVITY AND REPRODUCIBILITY

We train the Small baseline with three deterministic seeds. At step 8,000, mean training loss is 2.56487 ± 0.00234 and mean validation loss is 2.53432 ± 0.00186 (mean $\pm$ sample standard deviation, $n = 3$), with ranges 0.00447 and 0.00331. We also independently restart the seed-0 configuration from step 0 and repeat 2,000 steps. The two step-2,000 losses are 3.026079 and 3.027709, an absolute difference of 1.63×10^{-3}. Across 201 jointly logged points, the median/mean absolute differences are 5.47×10^{-3}/7.48×10^{-3} (maximum 6.33×10^{-2}). We therefore claim close statistical/trajectory reproducibility, not bitwise identity.

Table 10: Small baseline seed sensitivity and independent restart.

Run	Final train loss	Val. loss
Seed 0	2.566718	2.535311
Seed 1	2.562244	2.532176
Seed 2	2.565656	2.535484
Mean $\pm$ std.	2.564873 ± 0.002338	2.534324 ± 0.001862
Seed 0 original/restart @2k	$3.026079/3.027709$	–

The experiments use explicit scale-specific model configurations, deterministic seeds, fixed effective batch size 256, sequence length 4,096, and the optimizer/scheduler settings reported in Section 4. Evaluation is pinned to `lm-eval==0.4.12` at commit `6d642546f4688648fced259eb3302efd36ece5af`, with benchmark datasets obtained through the library's Hugging Face task loaders.